Architectures for Adaptive Weight Calculation on ASIC and FPGA

R L Walke, R W M Smith
DERA (Malvern), St. Andrews Road, Malvern, WR14 3PS, UK.
G Lightbody
DSiP™ Laboratories, Queens University of Belfast, Northern Ireland, UK.
email: walke@signal.dera.gov.uk

Abstract

We compare two parallel array architectures for adaptive weight calculation based on QR-decomposition by Givens Rotations. We present FPGA implementations of both architectures and compare them with an ASIC-based solution. The throughput of the FPGA implementations is of the order 5-20 GigaFLOPS, making FPGA a viable alternative to ASIC implementation in applications where power consumption and volume cost are not critical.

1. Introduction

The process of adaptive beamforming enables the beam pattern of an array of antennas to be shaped to counter interference arriving from directions other than that of the wanted signal[1]. The subsequent performance benefits are often sufficient to justify multiple antennas and their receivers, and the technique is now commonly employed in radar, sonar and more recently communications systems. In radar applications the data-rate is of the order of MHz and the sub-process of calculating the adaptive weights for a particular environment can be a very computationally demanding task. Therefore, its efficient implementation is a worthwhile topic of investigation.

Recursive least squares by QR decomposition is a well established technique for solving the least-mean-squares problem at the heart of adaptive weight calculation for both beamforming and filtering applications[2]. Good numerical performance is achieved by performing the algorithm using Givens rotations which allows the use of reduced wordlength arithmetic for efficient implementation on both ASIC and FPGA. Furthermore, a highly parallel triangular array processor architecture, known as the *QR-array*, exists that allows a very-high throughput implementation to be achieved by employing a large number of processors[3].

In reality, the number of processors used by the QR-array solution is too high and the throughput far in excess of the system requirements. To address this either the array is mapped to a reduced number of time-shared processors in a linear array, or alternatively, the triangular architecture is maintained and the processors themselves are constructed from time multiplexed arithmetic units (e.g. digit serial arithmetic). In this paper we adopt the former approach and consider two different mappings from the triangular QR-array to a linear one. We adopt this approach to provide a greater range of throughput options and to avoid the inefficiency that may arise from time-multiplexing at the arithmetic operator level within the processor.

The QR-array requires two types of operation, often referred to as boundary and internal cell operations. The mappings can be differentiated in the way cell operations are mapped to either dual operation or to distinct single operation processors. As such, we refer to them here as *mixed* and *discrete* mappings respectively. Both offer 100% or close to 100% utilisation over a broad range of problem size.

The mixed mapping was proposed by Rader[4] and adopts CORDIC (COordinate Rotation by DIgital Computer)[5] operators to realise the functionality of both boundary and internal cell operations.

The discrete mapping was developed by the authors to enable a whole range of algorithms based on standard arithmetic operators to be employed[6]. With these algorithms the function of the cells is quite different and the discrete mapping allows processors to be optimised for one or other cell operation. Algorithm variants have been developed to avoid square-roots and divisions, reduce operation count and allow fixed-point arithmetic to be employed. We use the Squared Givens Rotation (SGR) variant[7] here as it offers low operation count and is square-root free. It requires floating-point arithmetic but we only require low mantissa wordlength.

In this paper we explore the differences between the two approaches by considering FPGA implementations of both. We start by giving a brief overview of adaptive beamforming in radar systems in section 2. In this section we also include some numerical simulation results to es-

0-7803-5700-0/99/$10.00©1999 IEEE

tablish wordlength requirements for our two algorithms, as wordlength is critical to the size of our implementations. A brief overview of the two mappings is presented in section 3, followed by their FPGA implementations in section 4. We contrast the two approaches in section 5 and compare with an ASIC approach. We present our conclusions in section 6.

2. Adaptive beamforming in radar systems

Figure 1 provides an overview of an adaptive beamformer in which the outputs of an array of antennas are combined in a way which places nulls in the direction of interference whilst maintaining a high gain in the direction of interest. The antenna outputs are down-converted to base-band frequencies and digitised. The weights, w_i, to create a particular beam are calculated over a block of input data and then applied to this data via an array of complex multipliers to give the beamformed output.

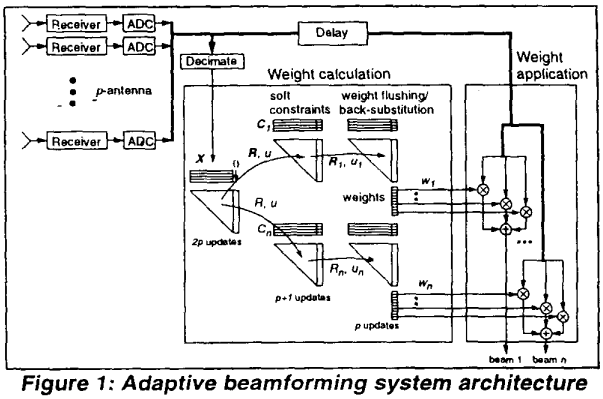

Figure 1: Adaptive beamforming system architecture

In the proposed system QR-decomposition is used to obtain the weights in a three-stage process. The first step is to decompose a minimum of $2p$ input samples of the input data, referred to collectively as $\boldsymbol{X}$, into an upper-triangular matrix $\boldsymbol{R}$ and vector $\boldsymbol{u}$. This captures the interference environment. To these are applied constraints for each beam, which stabilise the beam pattern and set a particular look-direction. The resulting $\boldsymbol{R}_i$ and $\boldsymbol{u}_i$ are related to the weights by $\boldsymbol{R}_i\boldsymbol{w}_i=\boldsymbol{u}_i$, and the weights are obtained by either back-substitution or weight-flushing (latter shown in figure). Back-substitution requires a separate processor but is computationally efficient, whereas weight flushing reuses the same hardware, but requires many more operations.

2.1 Wordlength requirements

The size of our FPGA or ASIC implementation is dependent upon the square of the wordlength (at least to a first approximation). Therefore, it is important to determine the minimum wordlengths from system simulations. In the adaptive beamformer application, the wordlength has to be sufficient to calculate the weights to an accuracy that allows full cancellation of the interference. To put this in perspective, the input contains 3 components: signal, interference and thermal noise (the latter from receiver components). The interference and signal are usually measured relative to thermal noise, where the signal is generally smaller and the interference much larger than thermal noise. The scaling of the ADC input is usually arranged so that thermal noise toggles the last bit or so, and has sufficient range to digitise interference without significant probability of saturation.

The task of the adaptive beamformer is to suppress interference to thermal noise levels. Therefore, the weights must be calculated with sufficient accuracy to do this. For 60dB interference (i.e. a voltage 1000 times greater than thermal noise), approximately 10-bits of accuracy are required to do this.

The accuracy of the weights depends upon the nature of the error and how it accumulates. Figure 2 shows the signal to interference+noise ratio (SNIR) for a range of wordlengths, for both the SGR and CORDIC algorithms. The number of antennas is 32 (i.e. p=32). At low wordlengths the SNIR is dominated by arithmetic errors, but as wordlength is increased the SNIR improves until it becomes dominated by the thermal noise (here, after post-processing, the maximum SNIR is 17dB) and there is no benefit in increasing the wordlength further.

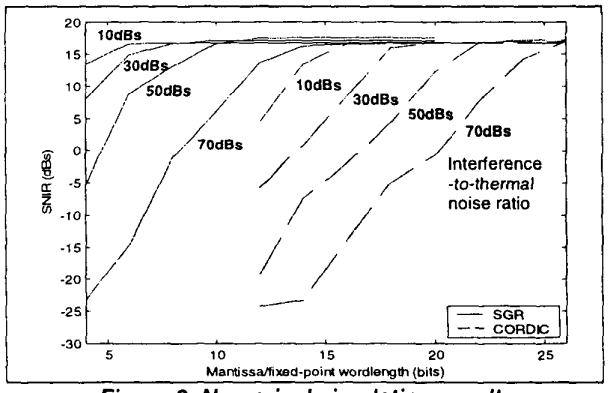

Figure 2: Numerical simulation results

In the following example implementations we choose a relatively high system performance of 70dB interferrence suppression and achieve this using 26-bit fixed-point wordlength for CORDIC, and 14-bit mantissa with 6-bit exponent for the SGR algorithm.

The wordlength requirements of CORDIC are much greater than those of the floating-point SGR algorithm for three reasons. Firstly, the fixed-point errors are generally greater than the floating-point errors. Secondly, there are more of them due to the use of sub-rotations and finally, and more significantly in this implementation, there is a small bias in the CORDIC errors. This latter fact results in

a more rapid growth of errors in the accumulated $\boldsymbol{R}$ and $\boldsymbol{u}$ terms (it grows with the number of operations n rather than $\sqrt{n}$ as in the floating-point SGR implementation). The larger the problem size, p, the greater this error. Rounding has been employed within the CORDIC operations to reduce error and bias, however some still remains.

3. QR-array mappings

3.1 QR-array

Figure 3 shows a 7 input QR-array (i.e. $p = 6$)[3] utilising $p(p+3)/2=27$ processors. The operation of the two processor types is shown in the insets. The antenna data enters from the top and progresses down the array. On each row the leading term is eliminated by a 2-d rotation between cell input and the elements of $\boldsymbol{R}$ and $\boldsymbol{u}$ which are stored within each cell.

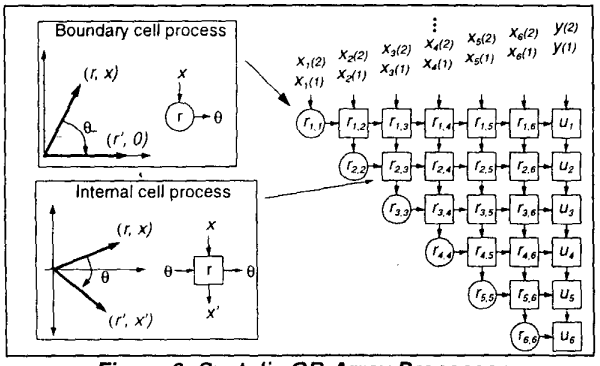

Figure 3: Systolic QR-Array Processor

The number of operations rapidly grows with problem size. However, techniques for mapping large arrays of operations onto a reduced number of processors are well established [8][9]. We consider two mappings which have been derived for the QR-array.

3.2 Mixed mapping

Figure 4 shows how the operations of the triangular array may be mapped onto dual function processors that are fully utilised. This is done by first moving the bottom portion of the array to give the same number of operations on each row in the resulting array. The operations are then allocated onto a linear array of processors. In this case two rows of operations are mapped onto each processor giving a sparse solution employing only 2 processors. Each processor performs the same number of operations (in this case 18) and so is fully utilised.

The order of execution of the operations on each processor is along each row from left to right, and rows top to bottom. Note that relocated operations will compute the tail-end of earlier QR-updates (to provide time for x-values to propagate down and then back up the processor array), with the effect that QR-updates are interleaved.

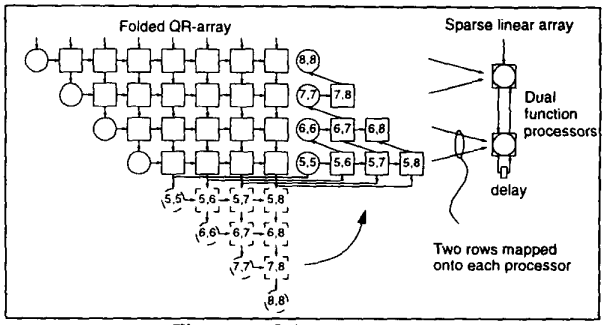

Figure 4: Mixed mapping

3.3 Discrete mapping

The need for dual-function processors is avoided by the discrete mapping shown in Figure 5. The array is obtained by first moving the lower portion of the array to the top, and then folding it to interleave operations. This places the same number of operations in each diagonal and ensures that local processor inter-connections are obtained when the operations are projecting down the diagonal onto a linear array of processors, as shown in Figure 5.

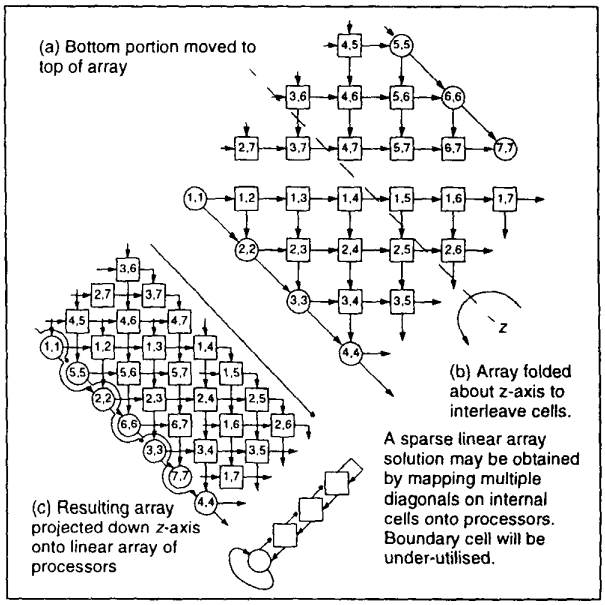

Figure 5: Discrete mapping

As with the mixed mapping, the relocated operations will finish the tail-end of an earlier QR-update.

4. FPGA implementations

We generate FPGA implementations of the two architectures from a hierarchy of structural VHDL descriptions. These are parametrised for wordlength, and include attributes to provide placement information to achieve very dense layout with predictable timing.

4.1 CORDIC - mixed mapping

CORDIC implements a 2-d rotation directly using a sequence of sub-rotations which can themselves be implemented by a sequence of shift and add/subtract operations. This maps very well onto Xilinx Virtex FPGAs as they implement fast carry-propagate adder/subtractors very efficiently. The carry propagation delay is so small relative to other delays that there is little speed advantage in using more complex redundant arithmetic adders such as signed-binary or carry-save.

Rader[4] showed how complex Givens rotations, can be implemented using 3 CORDIC rotations as shown in Figure 6. (See [5] and [4] for CORDIC operator details).

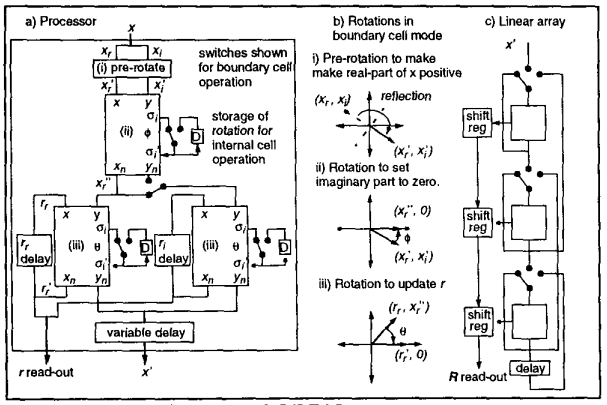

Figure 6: CORDIC processor

The processor is switched between boundary and internal cell operation as dictated by the schedule. In boundary cell mode the first CORDIC rotates the vector formed from the real and imaginary parts of the input, x, to set its imaginary part to zero. The real part is now processed as per Figure 3 i.e. it is rotated to zero against the stored r-term, and hence there is no x output in this mode. This requires only one θ-CORDIC, but the rotation is repeated in the imaginary datapath to generate rotation controls which can be stored locally for speed. The pre-rotation, ϕ-rotation and rotation are then repeated in subsequent internal cell operations, in which both θ-rotation CORDICs are required as the imaginary part of r may now be non-zero.

Each CORDIC output must be scaled by a constant $1/K$ to give a true circular rotation. This is avoided in our implementation by scaling the array input, x_{in}, up by an additional factor K on each iteration (i.e. K^n on the n-th iteration) to match the CORDIC scaling on the $\boldsymbol{R}$ and $\boldsymbol{u}$ terms. No correction is required on the x terms as the scaling is the same on each row, and so makes no difference to the final weights. To avoid overflow of both stored and x terms an occasional 1-bit shift-right operation is applied to the CORDIC outputs. The input scaling and shift control signals are pre-computed for a particular block length and stored in RAM.

Figure 7 shows the layout of a 4 processor array. Clearly visible are the 3 CORDIC blocks employed by each processor. The input scaling multiplier has not been included here, and would be combined with other pre-processing. The maximum clock rate is 108 MHz.

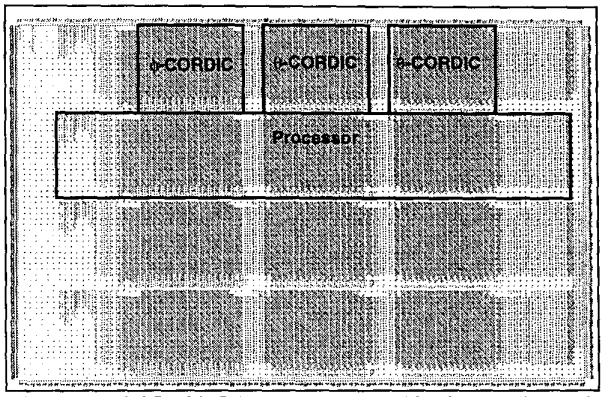

Figure 7: CORDIC QR processor on XCV1000BG560-6

4.2 Squared Givens Rotations - discrete mapping

Figure 8 shows a signal flow graph for the boundary and internal cell operations with the SGR algorithm.

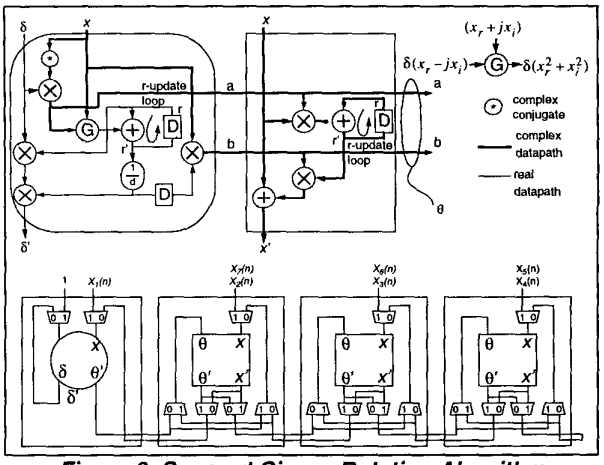

Figure 8: Squared Givens Rotation Algorithm

The loop to update the $\boldsymbol{R}$ and $\boldsymbol{u}$ quantities consists of a simple adder. This has two advantages. Firstly, the word-length of the adder can be increased to improve the accuracy to which $\boldsymbol{R}$ and $\boldsymbol{u}$ are accumulated over long runs of data. Secondly, with appropriate input scaling the adder can be made fixed-point and $\boldsymbol{R}$ and $\boldsymbol{u}$ terms updated on every clock cycle (assuming a processor for each cell in the array i.e. the QR-array is used). Therefore, very high sample-rate operation in excess of 100MHz is possible.

Figure 9 shows the implementation of 1 boundary and 2 internal cell processors on a Virtex FPGA. All operators are fully parallel and pipelined, accepting new operands on

each clock cycle. The maximum clock rate is 120MHz.

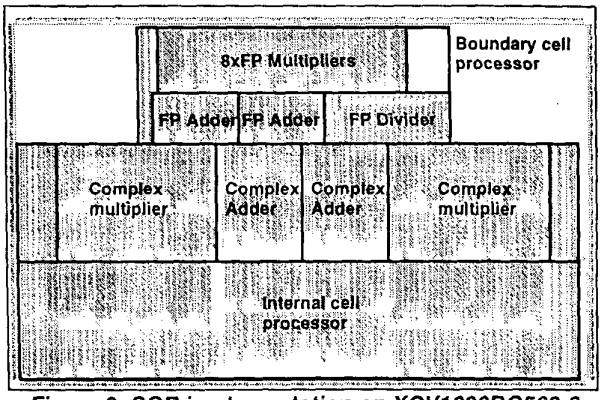

Figure 9: SGR implementation on XCV1000BG560-6

As with the mixed-mapping CORDIC implementation, larger arrays can be achieved, in this case, by implementing further internal cells on additional FPGAs interconnected in a linear array fashion. The input and output capabilities of current FPGAs easily support the required interconnect in this configuration. Additional pipelining delays can be included to move data between FPGAs and maintain full clock speed.

5. Comparison

The mappings and their implementations may be contrasted on a number of points:

*a) **Throughput***: This can be expressed in terms of number of actual or equivalent floating-point operations performed per second (FLOPS). Table 1 shows this measure calculated on the basis that boundary and internal cell processors employ 11 & 16 floating-point operations respectively (as for the SGR algorithm). The ASIC estimates are based on 0.35μm technology. This is not state-of-the-art, and we have scaled the throughput by a factor of 6 to obtain estimates for 0.18μm technology. However, the comparison with 0.35μm is useful to us, as the ASIC technology is more representative of what we would likely use given development timescales and budgets.

Table 1: Maximum throughput in FLOPS

FPGA	XCV1000-6 (0.22μm)			Estimated for XCV3200E (0.18μm)		
Algorithm	Clock (MHz)	Processors	MFLOPS	Clock (MHz)	Processors	MFLOPS
CORDIC	*108*	*4*	*6,372*[a]	*135*	*12*	*25,245*
SGR	*120*	*3*	*5,160*[b]	*150*	*9*	*20,850*
	ASIC, 0.35μm			ASIC, 0.18μm		
SGR	*100*	*21*	*32,900*	*190*	*74*	*225,000*

a. CORDIC is fixed-point. This is the equivalent number of FLOPS.
b. Results based on 100% boundary cell utilisation. For small numbers of processors and large problem sizes the boundary cell processor will be underutilised. For example, if $p=16$ then actual throughput would be 4,170 MFLOPS.

*b) **Weight update period***: A more useful interpretation of throughput to the system designer is the weight update rate. This is summarised in Table 2 for a range of problem size p, and is based on $2p$ data samples, followed by $p+1$ constraint inputs for each beam.

Table 2: Weight-update period estimates

		Weight-update period (μs)							
FPGA		XCV1000BG560-6				Future: XCV3200E			
Beams	Algorithm	Number of antennas (p)				Number of antennas (p)			
		16	32	64	128	16	32	64	128
1	***CORDIC***	*19.1*	*122*	*994*	*7,415*	*19.1*[a]	*48.9*	*283*	*2,224*
	SGR, FPGA	*27.8*	*213*	*1,673*	*13,244*	*5.55*	*42.7*	*335*	*2,649*
	SGR, ASIC	*8.33*	*32.0*	*251*	*1,987*	*N/A*			
3	***CORDIC***	*31.5*	*205*	*1,579*	*12,384*	*31.5*	*82.1*	*474*	*3,715*
	SGR, FPGA	*45.9*	*359*	*2,799*	*22,119*	*9.18*	*71.7*	*560*	*4,424*
	SGR, ASIC	*13.8*	*53.8*	*420*	*3,318*	*N/A*			

a. Rate limited by r-loop delay (discussed later), null operations inserted.

*c) **Latency:*** The time between applying the input and completing an update of ***R*** & ***u*** depends upon processor latency and the mapping. In Table 3 we summarise the latency in the result for a range of problem size.

Table 3: Latency in obtaining the R and u matrix

Algorithm	Clock (MHz)	Processor latency	Latency (μs) Number of antennas (p)			
			16	32	64	128
CORDIC[a]	*108*	*46*	*8.6*	*61.6*	*160*	*1,244*
SGR, FPGA	*120*	*55*[b]	*14.2*	*28.9*	*58.2*	*117*
SGR	*100*	*17*	*5.27*	*10.7*	*21.6*	*43.4*

a. Figures given for 4 processors.
b. Latency could be decreased to 39 by designing a 3 input adder.

In the mixed mapping, the data-flow is both down and up the array. The latter is against the order of execution of the processors and can introduce large latencies. Further latency is introduced when the outputs from one row of cell operations are not produced in time for the scheduled operations. In this case the processor outputs must be delayed until the next sequence of operations and the latency is increased accordingly.

The latency of the discrete mapping does not depend upon the number of processors. It is slightly larger for small problems size, p, but grows more slowly with p.

*d) **I/O requirements:*** For multi-chip implementations the number of interconnections is of interest. Table 4 summarises the number of bits required per chip. The discrete mapping has a relatively high I/O requirement and for the linear array it would be necessary to transmit 8*2*20=320 bits on every clock cycle (i.e. 4.8GBytes/s). This is well within the bounds of latest FPGAs. Bandwidth is reduced when sparse linear arrays are employed.

Table 4: I/O requirements of multi-chip linear array

Algorithm	Bus size	Core I/O	Input x	R,u readout	Total bits
CORDIC	26	2*2*26 (x)	uses core I/O	2*26	156
SGR	20	6*2*20 (θ,x)	2*2*20	2*20	360

e) Maximum throughput: This is achieved using the maximum number of processors. The full QR-array offers this, but requires that $\boldsymbol{R}$ and $\boldsymbol{u}$ matrices be updated on every clock cycle and therefore only a single clock cycle delay is allowed around the $\boldsymbol{R/u}$ update path. If this loop delay is greater than this then the number of processors that can be employed is reduced. Loop delay is dependent upon the latency of the operations used to update $\boldsymbol{R}$ & $\boldsymbol{u}$. The maximum number of processors is summarised in Table 5 for a range of problem size.

Table 5: Maximum number of processors

		Maximum number of processors				
Algorithm	Adder	Delay	4	8	16	32
CORDIC	*N/A*	*22*	*1*	*2*	*7*	*24*
SGR	*Float*	*11*	*1+1*	*1+3*	*2+11*	*3+46*
	Fixed[a]	*1*	*3+6*	*7+28*	*15+120*	*31+496*
SGR, ASIC	*Float*	*3*	*3+2*	*3+10*	*5+60*	*11+165*

a. Wordlength of adder increased to maintain numerical performance and if necessary redundant adder can be used to maintain single-cycle delay.

The large loop delay of CORDIC can seriously limit the maximum number of processors that can be usefully employed on a problem. Using a fixed-point representation of $\boldsymbol{R}$ and $\boldsymbol{u}$ in either the FPGA or ASIC implementations of the SGR algorithm gives a single-cycle delay and completely avoids this limitation.

f) Power Consumption: Power consumption is estimated to be of the order of 10-20W per chip for both ASIC and FPGA. Therefore, an estimate of throughput per Watt can be derived from throughput per chip (i.e. Table 1). The ASIC solution offers a factor of between 5-10 better throughput/Watt over FPGA.

6. Conclusions

The latest FPGAs offer a means to meet the performance requirements of adaptive beamforming systems without adopting ASICs, providing low volume cost and low power consumption are not required. With Virtex XCV3200E, due out next year, it has been estimated that a performance of 20 GigaFLOPS should be possible. This is over an order of magnitude greater than programmable DSP.

The rate of improvement in FPGA technology is staggering. Like DRAMs they take full advantage of the massive number of transistors offered by current fabrication technology, and these can be exploited by suitable parallel algorithm implementations. For DSP implementation we believe there is the potential for a further ten-fold improvement in performance by architecture optimisation, which when combined with technology improvement is likely to yield capabilities far in excess of current architectures.

We are building a demonstration system based upon the mixed mapping and floating-point SGR algorithm. It has better numerical performance than our CORDIC implementation and uses well established floating-point operations. At present, it consumes more chip area, but looking to the future, has more scope for optimisation both at the algorithm and implementation levels. Also, floating-point cores are likely to become standard, freely available and updated as FPGA technology progresses. Therefore, our array designs should be more portable. There is also more acceptance of designs by system designers based upon conventional floating-point operations.

The SGR implementation has high inter-processor I/O. This is only an issue for multiple FPGA solutions and is supportable by current FPGAs. Furthermore, with future FPGAs, multi-chip solutions are only likely to be required in the most demanding of applications.

7. References

[1] A. Farina, *Antenna-Based Signal Processing Techniques for Radar Systems*, Artech House, 1991.

[2] S. Haykin, Adaptive Filter Theory, 2nd Edition, Prentice Hall, ISBN 0-13-013236-5, 1991.

[3] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays", *Proc. SPIE 298, Real-Time Signal Processing IV*, pp. 19-26, 1981.

[4] C. M. Rader, "VLSI Systolic Arrays for Adaptive Nulling", *IEEE Sig. Proc. Mag*, Vol. 13, No. 4, pp. 29-49, 1996.

[5] J. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. Electron. Comput.*, Vol. EC-8, pp. 330-334, 1959.

[6] G. Lightbody, R. L. Walke, R. Woods, J. McCanny, "Novel Mapping of a Linear QR Architecture", *Proc. ICASSP*, vol. IV, pp. 1933-6, 1999.

[7] R. Döhler, "Squared Givens Rotations", *IMA J. of Numerical Analysis*, Vol. 11, pp. 1-5, 1991.

[8] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, ISBN 0-13-942749-X, 1988.

[9] G. M. Megson, *An Introduction to Systolic Algorithm Design*, Clarendon Press, ISBN 0-19-853813-8, 1992.

8. Acknowledgements

This work was carried out as part of Technical Group 10 of the MoD Corporate Research Programme. I would like to acknowledge the work by Alex Jackson on the implementation of the CORDIC QR-array processor and Chris Booth on the floating-point divider.